XG Boost

Reasons to use this algorithm:
Gradient boosting model is on the rise in competitive data science competitions. The internet is full of examples where the top entries used nothing but boosting algorithms. The advantages with XGB is that it uses a regularized model therefore yields better results especially for structured data such as the one we are using here. In general XBG is fast, reliable and efficient algorithm which works almost perfectly out of the box.

Language and Library used:
I used R's 'xgboost' library along with 'data.table' for dataframes.

Hyperparmameters used:

Hyperparameter	Default Value	Parameter Value used in the model
nrounds: Number of rounds	2	630
objective: This defines the loss function to be minimized	reg:linear	binary:logistic
eta: Analogous to learning rate in GBM	1	1
gamma: A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.	1	1
max_depth: The maximum depth of a tree	6	7
subsample: Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting	0.5	0.8
colsample_bytree: denotes the fraction of columns to be randomly samples for each tree.	0.1	0.8
min_child_weight: internal parameter to define resulted trees.	5	1
base_score: the initial prediction score of all instances, global bias	0.5	median(train_data)

Interpreting Results:

In order to interpret how well the model worked, we can take a look into the xgb.importance functions.
In our model, the model importance can be shown using the following commands:

```
importance <- xgb.importance(feature_names = colnames(dtrain),  model = gb_dt)
head(importance)
```

```
> head(importance)
                Feature       Gain      Cover   Frequency
1:           ps_car_13 0.07690467 0.06999898 0.044136879
2: ps_reg_02 ps_car_13 0.03094357 0.04512123 0.032230820
3:           ps_ind_03 0.03010975 0.04508349 0.027626417
4:     ps_ind_05_cat_0 0.02728141 0.01303455 0.005231037
5:           ps_reg_03 0.02628067 0.02256930 0.024275283
6: ps_reg_01 ps_car_13 0.02598662 0.03634769 0.028852441
```

Gain is the improvement in accuracy brought by a feature to the branches it is on. The idea is that before adding a new split on a feature X to the branch there was some wrongly classified elements, after adding the split on this feature, there are two new branches, and each of these branch is more accurate (one branch saying if your observation is on this branch then it should be classified as 1, and the other branch saying the exact opposite).

From the above, we can see that as we progressed through features, the gain has came down.

We can also plot the top 10 important features in our model:

```
> xgb.plot.importance(importance_matrix = importance[1:10,])
```